Generalized Connectionist Associative Memory

Nigel.Duffy' and Arun Jagota
Department of Computer Science
University of California
Santa Cruz, CA 95064
USA

Abstract

This paper presents a generalized associative memory model, which stores a collection of tuples whose components are sets rather than scalars. It is shown that all library patterns are stored stably. On the other hand spurious memories may develop. Applications of this model to storage and retrieval of naturally-arising generalized sequences in bioinformatics are presented. The model is shown to work well for detection of novel generalized sequences against a large database of stored sequences, and for removal of noisy black pixels in a probe image against a very large set of stored images.

1 Introduction

Associative Memory is an important problem, with numerous applications in AI and in pattern analysis. Here we consider this problem in the setting of recurrent connectionist networks. In this setting, the problem is defined as follows [Hopfield, 1982; Hertz *et* al., 1991; Kamp and Hasler, 1990]. Store m vectors $\mathbf{x}^1, \ldots, \mathbf{x}^m$, each in $\{0,1\}^n$, in the fixed points of a recurrent neural network in such a way that

1. most if not all the vectors $\mathbf{x}^p$ are stable, i.e. fixed points of the network.
2. the stable vectors $\mathbf{x}^p$ have reasonably large basins of attraction around them.

A lot of work has been done on this problem [Hertz *et al.,* 1991; Kamp and Hasler, 1990]. The problem has also been extended to store (i) real-valued vectors in $[0, 1]^n$ [Hopfield, 1984; Farrell and Michel, 1990] and (ii) vectors in $\{0, 1, \ldots, q-1\}$ for any integer $q > 2$ [Rieger, 1990; Kohring,; Jagota *et al,* 1998]. These have expanded the set of applications accordingly.

In this paper we present a further extension. Here we permit a component of a vector to store not merely a scalar value but a set of values. This expands the set of associative memory applications accordingly.

One illustrative application involves storing a collection of generalized consensus sequences (gsequences) that describe important signals, i.e. local regions with structure, in {D,R}NA or protein. Consensus sequences are commonly used to describe such signals [Haussler, 1998]. Generalized consensus sequences are a refinement in that not one (the most likely) but multiple (the top few) sites at a position are representable. Gsequences may also be used to describe multiple alignments of biosequences. Multiple alignments are an important procedure for capturing the similarities and differences between several related biosequences [Altschul, 1998].

Consider the problem of testing whether a given biosequence x matches any one of *m* given gsequences (representing signals or multiple alignments). One may store the gsequences in the generalized memory, then input x as a probe, and hope that the appropriate gsequence is recalled.

Applications of a similar nature arise in other settings.

Formally, the extended problem is defined as follows. We are given a positive integer *N* and fixed alphabets $\alpha_1, \ldots, \alpha_N$. A *gtuple* of length n $\leq$ AT is a tuple t of length *n* in which each component ti is an arbitrary subset of the alphabet α_i. A binary vector is a special case in which $\alpha_i = \{0,1\}$ for all $n = N$,n d each component of t is a singleton set (i.e., 0 or 1). The generalized associative memory problem is to store a given collection of m gtuples on given alphabets in such a way that

1. most if not all the gtuples $\mathbf{t}^p$ are stable.
2. the stable gtuples t^p have reasonably large basins of attraction around them.

Our main results are as follows. In this paper a recurrent neural network model for the generalized associative memories problem is proposed. The model is shown to have the stable storage property, i.e. all stored gtuples are necessarily fixed points of the network. Spurious fixed points may develop however. This is expected to make full retrieval performance poor on large stored collections. Here it is demonstrated that two limited types of retrieval—*novelty detection* and *noise removal*—work well even on large stored collections. With regards to the first, the following application is demonstrated. A large collection of generalized consensus sequences representing important multiple alignments of protein sequences is stored in the model. The model is shown to work well for detection of novelty in probe generalized sequences

against this stored set. With regards to the second, the following application is demonstrated. A very large collection of binary images is stored in the model by coding these images as gtuples. The model is shown to work well in removing noisy black pixels in noisy versions of stored images.

2 The Model

The model generalizes a model introduced earlier for the special case in which all gtuples are tuples (i.e., each component of a gtuple is a singleton set) [Jagota *et* al., 1998].

Let n_1 and n_2 be the minimum and maximum lengths of gtuples on alphabet α to be stored. Such gtuples will be called *library gtuples.* The model structure is described by a (partite) graph $G = (V,E)$ whose vertices represent neurons and edges their connectivity in the associated network. For convenience we describe all storage and retrieval operations on this graph; the correspondence to the connectionist network is established in a separate section. The vertex set V of G will be $(\alpha \times \{1,\ldots,n_2\}) \cup \{n_1,\ldots,n_2\}$. That is, V contains a vertex for every possible pair (alphabet-symbol, position) and for the various possible lengths $n_1,\ldots,n_2$. The edge set E of G will depend on the library gtuples $\mathbf{t}^1,\ldots,\mathbf{t}^m$. To describe the storage process the following notation is useful. For a given gtuple t of length n define n sets

$$V^j(\mathbf{t}) = \{(v,j) | v \text{ is an element of component } \mathbf{t}[j]\}$$

The sets $V^j(\mathbf{t})$ are subsets of the vertex set $V(G)$ of the graph G. Define the join $V^i(\mathbf{t}) \bowtie V^j(\mathbf{t})$ of two sets $V^i(\mathbf{t})$ and $V^j(\mathbf{t})$ as the set of edges joining every vertex in $V^i(\mathbf{t})$ with every vertex in $V^j(\mathbf{t})$.

The vertices in $V(G)$ are partitioned into $V^1, V^2, \ldots, V^{n_2}, \{n_1,\}, \ldots, \{n_2\}$ which we will call *columns.* These columns induce an $n_2 \times (n_2 - n_1 + 1)$-partite structure on G.

During our exploratory work we found that we were able to assure that the connectionist memory had the stable storage property only under the following restriction on the library gtuples. Let k = $(k_1,\ldots,k_{n_2})$ denote an n2-tuple of fixed positive integer values. We require that component i of every library gtuple contain exactly k_i elements fror α. Such gtuples are called k-uniform gtuples. Though this condition is very restrictive, it turns out that collections of arbitrary gtuples can be stored by recoding them to k-uniform gtuples for an appropriate choice of k (see below).

We now describe the storage process. The k-uniform library gtuples are presented sequentially, once each, to the storage algorithm. Initially, before any gtuple is presented, G has zero edges. When gtuple $\mathbf{t} = \mathbf{t}^p$ of length n *is* presented, the graph G is modified as follows.

- For each unordered pair $i \neq j$, $1 \leq i,j \leq n$, the edges in the join $V^i(\mathbf{t}) \bowtie V^j(\mathbf{t})$ are added to G (if some are already present, they are not added).
- For each i = 1, . . . , n, edges in the join $V^i(\mathbf{t}) \bowtie \{n\}$ are added to G if previously absent.

We now explain how arbitrary gtuples may be recoded to uniform ones. Given a collection $\mathcal{T}$ of gtuples, for each position $i = 1,\ldots,n_2$, let k_i denote the largest positive integer such that there exists a gtuple in $\mathcal{T}$ whose i_{th} component contains k_i values from the alphabet α_i. We now expand the alphabet α_i to include k_i more symbols $(i,1),\ldots,(i,k_i)$. We then recode each library gtuple as follows: whenever component j of library gtuple t^p contains $n_j < k_j$ values we add the values $(j,1),\ldots,(j,k_j - n_j)$ to this component. Under this recoding all library gtuples are now k-uniform.

From the storage rule, the structure of a stored gtuple as represented in the resulting graph G (after all the recoded gtuples have been stored) is readily apparent. This structure will lead to our definition of *memory.* For a given gtuple t of length n(t), define

$$V(\mathbf{t}) = \cup_{i=1,\ldots,n(\mathbf{t})} V^i(\mathbf{t}) \cup \{n(\mathbf{t})\}$$

that is, the set of vertices in G associated with t. Let $G[\mathbf{t}]$ denote the subgraph of G induced by V(t). If t is a library gtuple, $G[\mathbf{t}]$ has the following structure: for every $i \neq j$, $V^i(\mathbf{t})$ is joined with $V^j(\mathbf{t})$; for every $i = 1,\ldots,n(\mathbf{t})$, $V^i(\mathbf{t})$ is joined with the vertex labeled n(t). Furthermore, $|V^i(\mathbf{t})| = k_i$ for all i. We will call this structure a *k-uniform complete* (n(t) + *1)-partite subgraph* of *G.*

The structure of the previous paragraph leads to the following definition of memory, parametrized by the uniformity vector k. We will say that a subgraph *G[U]* of G induced by a vertex set $U \subseteq V(G)$ is a *memory* if and only if *G[U]* has the following structure:

1. U contains no more than one vertex from columns $n_2 + 1, n_2 + 2, \ldots$. If U contains such a vertex, call it u_n.
2. U contains no more than *ki* vertices from column $i \leq n_2$,
3. For all pairs $i \neq j, 1 \leq i,j \leq n_2$,, columns U^i and U^j are joined. Here $U^i = V^i \cap U$.
4. For all $i \leq n_2$, column U^i is joined with $\{u_n\}$.
5. U is a *maximal* vertex set satisfying conditions 3 and 4 under the constraints imposed by conditions 1 and 2.

The memory definition of the previous paragraph is attractive for the following reasons. First, the k-uniform complete $(n(\mathbf{t}) + 1)$-partite subgraph associated with a (recoded) library gtuple t satisfies this definition. Hence every (recoded) library gtuple is stored *stably* in the sense that it is necessarily recorded as a memory. Second, a Hopfield-type connectionist network may be associated with our graph G in such a way that this definition coincides with the fixed points of such a network. One consequence of this fact is that a memory is retrievable by simple hill-climbing. In the connectionist context it is also worth noting that, by contrast with our scheme, it is

impossible to store even three binary vectors stably in a binary Hopfield memory in the worst case [Dembo, 1989; Abu-Mostafa and Jacques, 1985].

Although all library gtuples are stored stably, spurious memories may develop. A spurious memory is a memory (see definition of memory given in an earlier paragraph) that is not associated with a library gtuple. It is the spurious memories that interfere with associative retrieval operations of the model. A good theoretical characterization of how spurious memories emerge in our model (as a function of the library gtuples) appears difficult. In this paper we indirectly assess the damage caused by spurious memories in various application contexts.

2.1 Illustrative Example

We now work through an example. Consider a multiple alignment of a set of DNA sequences of identical length.

ACCCAT
ACACAT
CCCCGT
TCCCAT

(With some loss of information) we describe this alignment by the generalized consensus sequence $\langle\{A,C,T\}, \{C\}, \{A,C\}, \{C\}, \{A,G\}, \{T\}\rangle$.

Consider now a collection of generalized consensus sequences of varying length.

$$\langle\{A,C,T\},\{C\},\{A,C\}\rangle, \quad \langle\{A,C\},\{G,T\}\rangle, \quad \langle\{A,C\}\rangle$$

To store this collection in our model we first determine k = (3,2,2). We then recode each gsequence to make it k-uniform in the manner described earlier, and then store the resulting collection according to the storage algorithm described earlier. Figure 1 shows the graph formed after all the recoded gsequences have been stored.

2.2 Connectionist Correspondence

The graph *G* formed after the recoded k-uniform gtuples have been stored may be represented as a connectionist (Hopfield-type) network in such a way that there is a one-to-one correspondence between the fixed points of the network and the memories of the graph (see the graph-structural definition bf memory in a previous section). The resulting connectionist network is an extension of the one in [Jagota *et al*., 1998].

The neurons in the network are the vertices in *G.* Symmetric weights between pairs of neurons are added as follows.

1. Each edge in the graph is represented by a weight of 0. network.
2. Each non-edge in the graph between two vertices not in the same column is represented by a sufficiently negative weight.
3. Each pair of vertices *i,j* in the same column is connected by a weight of - 1 .
4. Each pair of vertices i, *j* representing gtuple lengths is connected by a sufficiently negative weight.

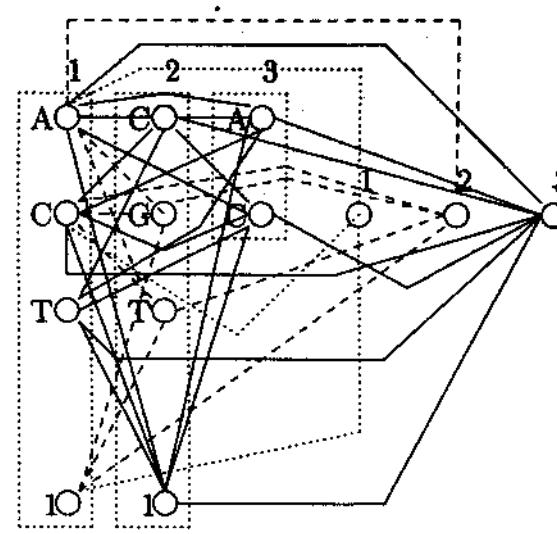

Figure 1: Graph formed after storing the recoded gsequences $(\{A,C,T\},\{C,1\},\{A,C\})$, $(\{A,C,1\},\{G,T\})$, and $(\{A,C,1\})$. Edges with solid, dashed, and dotted lines represent those added to the graph when the first, second, and third of the above gsequences is stored, in this order. This distinction is only for illustration sake; in the graph all edges are identical.

5.

All self-weights are 0.

Biases (constant external inputs) to neurons are as follows.

1. Each tuple-length neuron (neuron labeled n_2 + 1,...,) has a bias of 0.5.
2. Each neuron in column i (including the expanded-alphabet neurons in that column) has a bias of *ki* — 0.5.

This yields a Hopfield network with binary 0/1 neurons and a symmetric, zero-diagonal weight matrix, whose fixed points are the local minima of a certain energy function [Hopfield, 1982]. The one-to-one correspondence between these fixed points and the memories of the graph is easily established. The weight condition 2 ensures that in a fixed point if two neurons from different columns are ON, then the associated vertices are adjacent in the graph. The weight condition 3 ensures that in a fixed point at most one tuple-length neuron is ON. The bias condition 2 taken together with the weight condition 3 ensures that in a fixed point not more than *k{* neurons are ON in column i.

3 Results

As noted earlier, when a large number of library patterns are stored, the model is not expected to work

well on full retrieval of library patterns from distorted probe patterns. This is due to the compact architecture and distributed storage mechanism, which leads to a low error-correction capacity [Hertz *et al.,* 1991]. It turns out nevertheless that limited kinds of retrieval on large collections of stored library patterns is feasible. The first kind is *novelty detection* in which one wishes to test if a probe pattern is novel relative to the collection of library patterns. The second kind is *noise removed* in which one wishes to filter out noise from a probe pattern obtained by adding noise to a library pattern. The precursor of this model has been shown to work well on both these kinds of retrieval [Jagota, 1994; Duffy and Jagota, 1998]. Here we demonstrate that the generalized model works on these types of retrieval as well.

3.1 Novelty Detection

To demonstrate novelty detection, we picked a significant task from Bioinformatics. We start with a dataset of 3,202 blocks, each block representing a multiple alignment of segments of certain proteins, assembled by [Henikoff and Henikoff, 1991]. We convert each block to a generalized consensus sequence on the twenty-letter protein alphabet. The resulting collection of 3,202 gsequences is stored in the model. The average length of a gsequence in this collection is 32.56; the average size of the set in a position in a gsequence is 4.2.

To assess the model's performance we distorted each library gsequence by replacing each value *v* in the value-set stored at position *j* by a random alphabet-symbol with probability p. Figure 2 reports, as a function of p, the percent of the 3,202 probe gsequences that were detected as not in the library collection by the model. For $p = 0$, as expected, none of the probe gsequences was flagged as novel because all are the library patterns; hence stable. The percentage of probe patterns detected as novel increased as p was increased, which demonstrates that the model works well on this task.

It turns out that the model is also able to report (an efficiently-computable) score for each probe gsequence indicating its distance from the stored library collection. The following definition of score generalizes the one in [Duffy and Jagota, 1998]. For a vertex set $U = U(t)$ associated with a probe gsequence t define score(U) as the number of pairs of vertices in *U* that are in different columns and are non-adjacent divided by the number of pairs of vertices in *U* in different columns.

Figure 3 reports, as a function of p, the average score of the probe patterns. We see that the average score increases linearly with p. This validates the suitability of this scoring scheme.

3.2 Noise Removal

To demonstrate noise removal, we picked the following task from image processing. We stored in our model a dataset of 60,000 binary images of handwritten digits. Each image was roughly 28 x 28 pixels, with approximately 140 of them being black. To store an image in our model we coded it as a gtuple as follows. Each column of an image represented a position in the gtuple. The positions of the black pixels in column t in an image was the value-set stored in component *t[i]* of the associated gtuple.

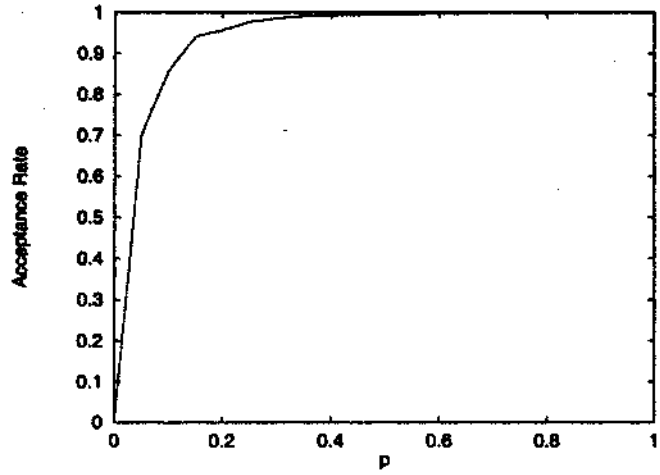

Figure 2: Percentage of probe gsequences detected as novel, as a function of the distortion probability *p*.

As our test set, we picked 3000 of these images and distorted them by converting each white pixel into a black pixel with probability 0.01. On average this added 6.4 pixels of black noise in the probe image. We then used a noise-removal algorithm in conjunction with our model that simply converts (sequentially) those black pixels to white pixels in the probe gtuple that are maximally non-adjacent to other black pixels. We found that on average this mechanism cleaned up 3.3 of the 6.4 noisy black pixels in the probe.

This task may also be mapped to the original connectionist model [Jagota, 1994], by treating each library image as a (28 x 28)-bit binary vector. However this has some drawbacks. The resulting network has twice as many neurons. More importantly, this coding is dense (half the neurons in the network participate in each memory). By contrast only about 1/6th of the neurons in the network (140 out of 28 x 28) participate in a memory in the generalized network.

4 Current and Future Work

One appealing feature of this model is that it may also be used to (efficiently) assign a score to a probe sequence—a type of "distance" from the entire stored library collection. This opens up some interesting application possibilities, that we are now beginning to investigate.

References

[Abu-Mostafa and Jacques, 1985]
Y.S. Abu-Mostafa and J.S. Jacques. Information ca-

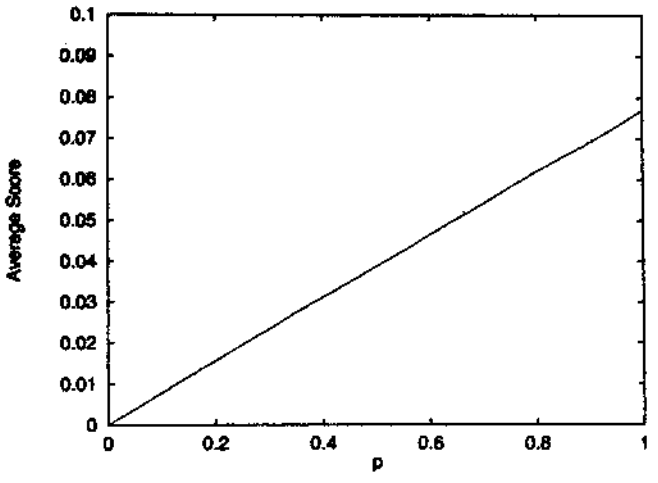

Figure 3: Average score of a probe gsequence as a function of *p.*

pacity of the Hopfield model. *IEEE Transactions on Information Theory,* 31(4):461-464, July 1985.

[Altschul, 1998] S.F. Altschul. Fundamentals of database searching. *Trends Guide to Bioinformatics,* pages 7-9, 1998. Elsevier TRENDS Journals, Trends Supplement 1998.

[Dembo, 1989] A. Dembo. On the capacity of associative memories with linear threshold functions. *IEEE Transactions on Information Theory,* 35(4):709-720, 1989.

[Duffy and Jagota, 1998] N. Duffy and A. Jagota. Connectionist password quality tester, 1998. Submitted.

[Farrell and Michel, 1990] J.A. Farrell and A.N. Michel. A synthesis procedure for Hopfield's continuous-time associative memory. *IEEE Transactions on Circuits and Systems,* 37(7):877-884, July 1990.

[Haussler, 1998] D. Haussler. Computational genefinding. *Trends Guide to Bioinformatics,* pages 9-12, 1998. Elsevier TRENDS Journals, Trends Supplement 1998.

[Henikoff and Henikoff, 1991] S. Henikoff and J. G. Henikoff. Automated assembly of protein blocks for database searching. *Nucleic Acids Research,* 19:6565-6572, 1991.

[Hertz *et al,* 1991] J. Hertz, A. Krogh, and R.G. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley, 1991.

[Hopfield, 1982] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences, USA,* 79, 1982.

[Hopfield, 1984] J.J. Hopfield. Neurons with graded responses have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences, USA,* 81, 1984.

[Jagota *et al.,* 1998] A. Jagota, G. Narasimhan, and K.W. Regan. Information capacity of binary weights associative memories. *Neurocomputing,* 1998. to appear.

[Jagota, 1994] A. Jagota. Contextual word recognition with a Hopfield-style net. *Neural, Parallel, & Scientific Computations,* 2(2):245-271, 1994.

[Kamp and Hasler, 1990] Y. Kamp and M. Hasler. *Recursive Neural Networks for Associative Memory.* Wiley, New York, 1990.

[Kohring,] G.A. Kohring. On the q-state neuron problem in attractor neural networks. *Neural Networks,* ??:??, ?? ??

[Rieger, 1990] H. Rieger. Storing an extensive number of grey-toned patterns in a neural network using multistate neurons. *Journal of Physics A,* 23.L1273-L1280, 1990.